Deep Poetry: Word-Level and Character-Level Language Models for Shakespearean Sonnet Generation

Stanley Xie
Department of Computer Science
Stanford University
Stanford, CA 94305
stanleyx@stanford.edu

Ruchir Rastogi
Department of Computer Science
Stanford University
Stanford, CA 94305
rrastogi@stanford.edu

Max Chang
Department of Computer Science
Stanford University
Stanford, CA 94305
mchang4@stanford.edu

Abstract

Text generation is a foundational task in natural language processing, forming the core of a diverse set of practical applications ranging from image captioning and text summarization to question answering. However, most of this work has focused on generating prose. We investigate whether deep learning systems can be used to synthesize poetry, in particular Shakespearean-styled works. Previous work on generating Shakespeare prose involved training models exclusively on the word or character level. Here, we implement those previous models for poetry generation and show that models that combine word and character level information, such as a Gated LSTM and a CNN-based LSTM, significantly outperform the baseline word-LSTM and char-LSTM models. Perplexity scores for the two complex models are almost 10 fold better than that for our baselines, and human ratings of the model-generated sonnets reflect this as well. In particular, the sonnets our complex models generate have a coherent meaning and relatively correct meter without blatantly copying Shakespeare's original works. These results encourage us that models that blend word and character level information would be useful for a variety of tasks outside of just poetry generation and may be crucial in bridging the gap between computer generated and human written text.

1 Introduction

"Poetry is language at its most distilled and most powerful." Rita Dove

In recent years, deeper, more powerful neural networks have outclassed previous computational models on a variety of natural language processing tasks, from dependency parsing to sentiment analysis. Despite their sweeping success, deep learning algorithms have yet to be applied effectively to poetry generation.

In this paper, we hope to take a first step in training deep learning models for poetry generation. Specifically, we focus on the task of generating Shakespearean poetry for two reasons. First, Shakespeare's works are widely recognized as a gold standard for poetry and language. Filled with cryptic and layered messages as well as complex language patterns, they have been heavily analyzed and

reinterpreted. Being able to understand Shakespeare requires a mastery of English, not just semantically (i.e. an appreciation of Early Modern English vocabulary) but syntactically as well (i.e. sentence structure, rhyming, meter). To be able to generate Shakespeare-like text represents an even higher level of understanding and mastery. Second, an amalgamation of Shakespeare's full works provides a sizable corpus for our model to learn from. No other poet was as renowned or as prolific as "The Bard" himself.

Shakespearean poetry comes in the form of sonnets. The structure of these sonnets is as follows: there are 14 lines in each sonnet, broken down into three quatrains (four line verses) and a rhyming couplet at the end. Each quatrain has an ABAB rhyming scheme, and each line follows the iambic pentameter structure. In this pattern, each line consists of five iambs (a stressed syllable followed by an unstressed syllable), resulting in each line having ten syllables.

2 Relevant Work/Background

Computational poetry generation is a difficult problem that many researchers have tried to tackle before. Some begin with a set of constraints on meter, word similarity, rhytm, etc. and attempt to use a corpus to satisfy these constraints [1]. Others have focused on generating poetry with a more emotional touch [2]. Generating rhyming and rhythmic patterns has also been a major focus. Recently, a "deep-rhyming" language model was created for rap lyric generation [3] and another paper focused on generating Chinese quatrains which also have strict rhyme and rhythmic schemes [4]. However, in both cases, these models did not organically capture the rhyme scheme and rhythmic patterns. Instead, these features were manufactured: they used a beam-search like methodology in which they generated all possibilities that conformed to the poetic structures, and then used neural networks to rank the possibilities and pick the top selection. In our models, we attempt to capture these patterns organically in an end-to-end trained model.

Previous work on generation of Shakespearean text comes from Karpathy in 2015 [5]. In the article, he discusses the effectiveness of character RNNs in generating texts of all sorts, ranging from Paul Graham's essays to Shakespeare.

3 Datasets

Our dataset comes from MIT's "Complete Works of William Shakespeare," which contains all of his plays/writings and 154 sonnets [6]. We split the sonnets into a training set, a development set, and a test set with 124, 15, and 15 poems respectively.

4 Models

4.1 Baseline Models

As previously mentioned, our work is inspired by Andrej Karpathy's use of a char-RNN. However before we discuss that model, we describe an even simpler one, a word-RNN. In this model, word embeddings are trained on the entire corpus of Shakespearen text, encompassing 975125 tokens, using GloVe [7]. These 300 dimensional word embeddings are then fed into a multiple-layer RNN where they are trained to predict the next word in Shakespearen sonnets. In essence, we use this model to train a language model on words that is more sophisticated than a naive bigram approach.

The second baseline model we generate is a replicate of Karpathy's char-RNN. The key distinction between this model and the previous one is that predictions are made on the character level as opposed to on the word level. The character embeddings are not preinitialized using GloVe but instead are also learned during training time.

For these two models, we experiment with the type of RNN used: vanilla RNN, GRU, and LSTM. We also tune the number of layers in our networks. However, there are a few things we do assume from the beginning: (a) word embeddings for the word-RNN are chosen to be 300 dimensional because in the original GloVe paper that was proven to optimize the tradeoff between performance in generic NLP tasks and the complexity of the network; (b) hidden layers have 512 dimensions, as optimized by Karpathy; and (c) dropout = 0.5 at each layer (also optimized by Karpathy).

4.2 Gated LSTM

The above baseline models are effective in a variety of language modeling tasks. However, the two models seem to capture distinct levels of information, both of which are valuable for poetry generation. Word-RNNs are trained on word-level representations and thus are quite effective at capturing semantic meaning. Char-RNNs are trained on character-level representations and thus potentially contain more information about morphemes and–more relevantly for our purposes–rhyming and meter.

Thus, the complex models we explore attempt to combine both word-level and character-level representations. The first model we implemented was a Gated LSTM heavily inspired from a recent paper by Miyamoto and Cho [8]. The architecture is illustrated in Figure 1.

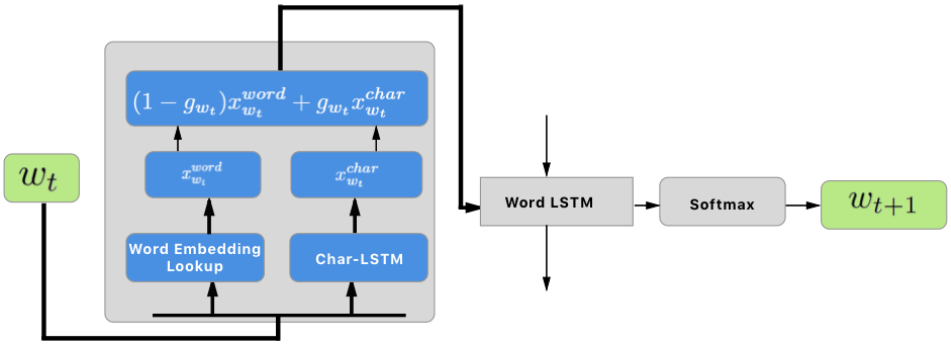

Figure 1: Architecture of the Gated LSTM

At each step, the most recent word w_t is taken, where w_t is a one-hot vector of length $|V|$ (the size of the vocabulary). From this, we generate two representations, one at the word level and the other at the character level: $x^{word}_{w_t}$ and $x^{char}_{w_t}$. Here, $x^{word}_{w_t}$ is simply the d-dimensional embedding of w_t created by running GloVe on the entire Shakespeare corpus.

To calculate $x^{char}_{w_t}$, we feed in the word–character by character–into an LSTM. (The hidden states are initialized using the final hidden states from the previous word.) After running each character through the LSTM, we transform the final hidden state as follows:

$$x^{char}_{w_t} = W_1 \dot{h_{w_t}}$$

where W_1 is a parameter of dimension $d \times h$ (d is the embedding length and h is the number of dimensions in the LSTM's final hidden state).

These two representations are joined as follows:

$$x_{w_t} = (1 - g_{w_t})\dot{x^{word}_{wt}} + g_{w_t}\dot{x^{char}_{w_t}}$$

Here, g_{w_t} is defined as such:

$$g_{w_t} = \sigma(v_g^T \dot{x^{word}_{w_t}} + b_g)$$

In the above equation, v is a vector $\in R^d$ and and $b_g \in R$. Both are learnable, adaptive parameters. The result x_{w_t} is then run through a LSTM with three layers. The output of the LSTM is transformed using softmax in order to generate the probability of the next word in the sequence.

4.3 CNN-based LSTM

The other model we implement is a CNN-based LSTM, originally described in a paper by Kim *et al.* [9]. Unlike in the Gated LSTM, word embeddings are not explicitly trained using GloVe. Instead, a CNN is used to generate a word embedding for each word w_t, and that embedding is then fed into an LSTM in order to generate a language model. The overall architecture is depicted in Figure 2.

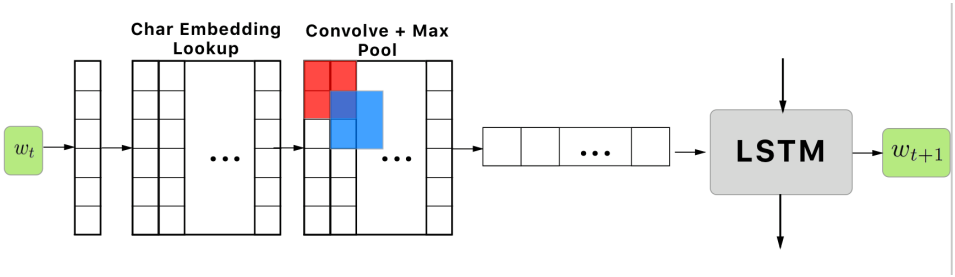

Figure 2: Architecture of the CNN-based LSTM

Since the LSTM portion here is the same as that for previous models, we focus on the generation of the word embedding using the CNN. For each word, we split it apart into individual characters and feed into the CNN a concatenation of the character embeddings (which are not pretrained but also learned). This concatenation outputs a matrix E of size $n \times d$ where n is the number of characters in the word and d is the number of dimensions in each character embedding.

On this matrix E, we then run a convolutional layer using filters of size $k \times d$. We choose k to be of various sizes $[2, 3, 4, 5]$ and had 25 filters of each size. These filter sizes were chosen because they capture information about words at the morpheme level and generally reflect the length of syllables. After this convolutional layer, we apply a max pool on each filter in order to hone in on only the important information that each filter is picking up. Since there are 100 filters in our model, the CNN at the very end outputs a 1×100 vector that reflects the word embedding for w_t. That word embedding is then fed into the LSTM as normal to predict the next word.

5 Methods

We quantify the goodness of the poems we generate using both quantitative and qualitative measures. As is common in language modeling tasks, we measure the perplexity (PPL) of our model using the test set in order to capture the likelihood of our model to correctly predict the actual next word in Shakespeare's sonnets. Formally, perplexity is defined by the following formula:

$$PPL = 2^{\frac{NNL}{T}}$$
$$= 2^{-\log \frac{1}{T} \sum_{t=1}^{T-1} P(w_{t+1}|(w_1 \cdots w_t))}$$

In the above equations, NNL denotes negative log likelihood, and T corresponds to the length of the sequence on which we are making predictions.

However, using only a computer to rate poetry fundamentally misses the point. Poems speak to the human existence and thus require actual human evaluation. As such, we asked friends to evaluate sonnets our models generated on three measures using a 1-5 scale: (1) coherence, (2) poeticness, and (3) meter/rhyme. 8 participants were surveyed, and each was asked to evaluate three quatrains from each model. Participants were not told which poems were generated by which models, in order to reduce the potential of any bias.

6 Results

6.1 Hyperparameter Tuning

We first describe the results of the hyperparameter tuning experiments. As mentioned previously, we chose not to tune some hyperparameters since they had previously been optimized by other researchers. Instead, we focused on two broad questions: (a) "which RNN variety should we use?" and (b) "how many layers should our RNN contain?"

To answer the first question, we measured the perplexity of the word-RNN with three different RNN varieties: vanilla RNN, GRU, and LSTM. The LSTM had a much lower perplexity (around 12) than

both the vanilla RNN (around 35) and the GRU (around 30). This is not particularly surprising since LSTM's do a much better job than the other two at maintaining long-term sequence memory, which is particularly important for poetry generation given the abundance of extended motifs and metaphors in poems. Due to the strikingly good results of the LSTM, we decided to use them in all other models as well.

To determine the optimal depth of the LSTM, we determined how the perplexity of a word-LSTM changed depending upon the number of layers. Figure 3 depicts the results:

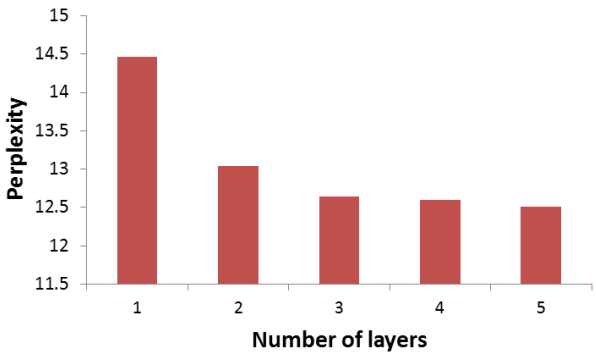

Figure 3: Perplexity in a word-LSTM model as a function of the number of layers

The perplexity of the word-LSTM model decreases until there are three layers, at which point the perplexity flattens out. Thus, we designed all models with three layers in the LSTM in order to decrease the perplexity of the model without adding unnecessary parameters that might encourage overfitting.

6.2 Quantitative Results

To determine the goodness of each model, we measured their perplexity on the test set. Table 1 displays the results:

Table 1: Average perplexity of each model (n = 15)

Model	Word-LSTM	Char-LSTM	Gated LSTM	CNN-based LSTM
Perplexity	12.644	19.526*	1.170	2.349

Clearly, the Gated LSTM and the CNN-based LSTM outperform the naive word-LSTM by almost 10 fold. However, it is impossible to compare the char-LSTM to any of the other models. That is because the char-LSTM is evaluated on its ability to correctly predict the next character, whereas all other models are evaluated on their ability to predict the next word. These are fundamentally incomparable scales. Thus, we must resort to qualitative results to determine the goodness of the char-LSTM.

Given the extremely low perplexity of our complex models, we were wary that our models were just spitting out Shakespeare word for word and not actually synthesizing new poetry. In order to test if this was occurring, we counted the number of trigrams (bigrams provide extremely noisy data) generated by each model that are also present in Shakespeare's sonnets. For all models, the percent of "plagiarized" trigrams (i.e. a trigram generated by the model that is also present in Shakespare's sonnets) is fairly low, as shown in Figure 4. This is especially true for the char-LSTM and the Gated-LSTM. However, the char-LSTM potentially has artificially low % plagiarism since it oftentimes misspells words, preventing potential "plagiarized trigrams" from being identified. It is also noteworthy that the Gated-LSTM has a much lower plagiarism rate than the CNN-based LSTM even though both have very similar perplexity scores.

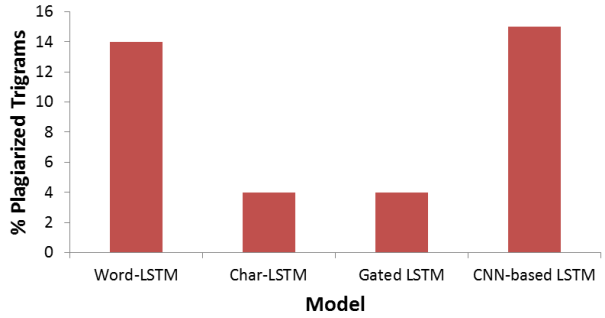

Figure 4: Uniqueness of sonnets generated (n = 5)

Moreover, we ran experiments to test how well the model-generated sonnets conformed to the iambic pentameter meter scheme. While it is unreasonable to expect a purely text-driven system to conform to the pattern of stressed syllables followed by unstressed syllables that is present in iambic pentameter, it is possible for these models to have learned that each line should have exactly 10 syllables. To test this, we hand-counted the number of syllables per line generated by the four models. The results are illustrated in Figure 5.

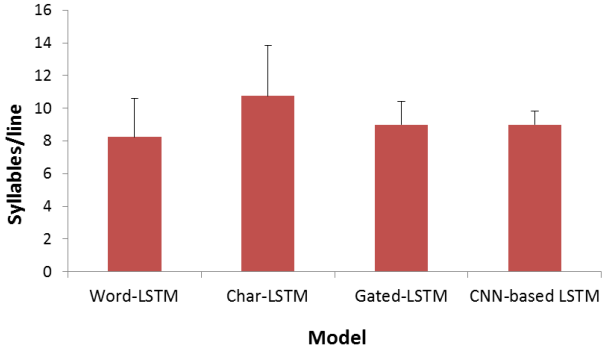

Figure 5: Number of syllables per line for each model (n = 5 and error bars indicate 1 std)

Surprisingly, all models seem to do a fairly decent job at generating text with 10 syllables per line. The much smaller error bars for the Gated-LSTM and the CNN-based LSTM suggest that both word and character-level information is necessary for obtaining the correct meter, contradicting our hypothesis that only character-level information would be important.

6.3 Qualitative Results

Quantitative results provide just half of the story, though. Survey participants' responses, shown in Table 2, allow us to understand how people actually perceive computer generated sonnets.

Table 2: Survey results (n = 8). All categories are on a 1-5 scale

Model	Coherence	Poeticness	Meter/Rhyme	**Average**
Word-LSTM	2.25	2.63	2.00	2.29
Char-LSTM	1.50	1.88	2.00	1.79
Gated LSTM	4.00	3.63	3.25	3.63
CNN-based LSTM	3.75	3.50	3.25	3.50

The survey results correspond quite nicely to the perplexity scores. Participants rated sonnets generated by the Gated LSTM and the CNN-based LSTM much more highly than those generated by the baseline model. In particular, the coherence of those sonnets seems to drive people's affinity towards the Gated LSTM and CNN-based LSTM models. Additionally, the higher scores in "Meter/Rhyme" category for the complex models seems to be a direct reflection of their limited variance in syllables per line, as rhyming is almost non-existent in all sonnets generated.

We also leave here a randomly sampled quatrain from each of the four models, so that the reader may judge the quality of each for themselves.

Word-LSTM
To me, fair, so you never be,
Each trifle, way, when bore your beauty when,
Such hence your can still,
O thou how much were your self the wrong chide

Char-LSTM
Give body an my mistress, trease,
Pert do I blind no contenden old?
In a mappess, can my mistress'd to brand
In a sovereigh I canst turns more desire to cruent,

Gated LSTM
Thy youth 's time and face his form shall cover?
Now all fresh beauty, my love there
Will ever Time to greet, forget each, like ever decease,
But in a best at worship his glory die.

CNN-based LSTM
For that deep wound it gives I loved me not;
That I might see me, than now are thief;
That which it might me, and words old tell
That I are one with my friend?

7 Conclusion and Future Work

Overall, through this project, we have shown that neural network models that combine both word and character information to generate Shakespearean poetry significantly outperform word-LSTM and char-LSTM models that exclusively look at only of of the two facets. This is confirmed not only by the models' perplexity scores but also by human feedback to the model-generated sonnets. In particular, the massive increase in the coherence of the sonnets generated by the complex models is quite encouraging because it suggests that these models might have broader applicability. Combining word and character representations is not a technique that should be limited to the obtuse and relatively impractical field of poetry generation. It should be commonplace in all sorts of various natural language programming tasks, particularly in those that involve human-computer communication.

However, this project is interesting in and of itself, and in the future, given sufficient time we would like to try out a few more ideas. First, we hope to incorporate attention into our model to help us produce better rhyming throughout the poem (a goal that we clearly did not meet with our current models). We believe that attention will be particularly helpful for this task because the rhyme scheme of Shakespearean sonnets is very rigidly defined, so it's very easy to pinpoint which word the current word should be rhyming with. Moreover, we believe it is possible to combine the Gated LSTM and the CNN-based LSTM to produce an even more powerful language model learner. The CNN seems to capture morpheme level information that is currently not present in the Gated LSTM, as it only receives word-level and individual character-level information. Having the word embedding output by the CNN be a third factor that the Gated LSTM must consider would probably significantly help with meter and rhyme.

8 Contributions

Stanley Xie wrote most of the word-LSTM and the char-LSTM. Max Chang wrote most of the Gated-LSTM. Ruchir Rastogi wrote most of the CNN-based LSTM. All members in the group contributed equally to the poster and the final paper.

References

[1] Colton, S. (2017). Full-FACE Poetry Generation. [online] Available at: http://ccg.doc.gold.ac.uk/papers/colton_iccc12.pdf

[2] Misztal, J. and Indurkhya, B. (2017). Poetry generation system with an emotional personality. 1st ed. Computational Creativity.

[3] Jones, M. (2017). DeepRhyme (D-Prime) generating dope rhymes with deep learning. [online] Available at: https://swarbrickjones.wordpress.com/2016/11/07/deeprhyme-d-prime-generating-dope-rhymes-with-deep-learning/

[4] Wang, Z. (2017). Chinese Poetry Generation with Planning based Neural Network. [online] Available at: https://arxiv.org/pdf/1610.09889.pdf

[5] Karpathy, A. (2017). The Unreasonable Effectiveness of Recurrent Neural Networks. [online] Available at: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[6] The Complete Works of William Shakespeare. [online] Available at: http://shakespeare.mit.edu/

[7] Penington, J., Socher, R., and Manning, C. (2014). GloVe: Global Vectors for Word Representation. [online] Available at: https://nlp.stanford.edu/pubs/glove.pdf

[8] Miyamoto Y. and Cho K. (2016). Gated Word-Character Recurrent Language Model. [online] Available at https://arxiv.org/abs/1606.01700

[9] Kim, Y. *et al.* (2015). Character-Aware Neural Language Models. [online] Available at https://arxiv.org/abs/1508.06615